ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

Multi-Objective Optimization of HPC Kernels for Performance, Power, and Energy

Prasanna Balaprakash[*], **Ananta Tiwari**[†], **Stefan M. Wild**[*]

Mathematics and Computer Science Division

Preprint ANL/MCS-P4069-0413

April 2013

http://www.mcs.anl.gov/publications/

[*] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA
[†] Performance Modeling and Characterization (PMaC) Lab, San Diego Supercomputer Center, La Jolla, CA 92093, USA

*

Multi-Objective Optimization of HPC Kernels for Performance, Power, and Energy

Prasanna Balaprakash
Mathematics and Computer Science Division
Argonne National Laboratory
pbalapra@mcs.anl.gov

Ananta Tiwari
Performance Modeling and Characterization (PMaC) Lab
San Diego Supercomputer Center
tiwari@sdsc.edu

Stefan M. Wild
Mathematics and Computer Science Division
Argonne National Laboratory
wild@mcs.anl.gov

ABSTRACT

Code optimization in the high-performance computing realm has traditionally focused on reducing execution time. The problem, in mathematical terms, has been expressed as a single-objective optimization problem. The expected concerns of next-generation systems, however, demand a more detailed analysis of the interplay among execution time and other metrics. Metrics such as power, performance, energy, and resiliency may all be targeted together and traded against one another. We present a multi-objective formulation of the code optimization problem. Our proposed framework helps one explore potential tradeoffs among multiple objectives and provides a significantly richer analysis than can be achieved by treating additional metrics as hard constraints. We empirically examine a variety of metrics, architectures, and code optimization decisions and provide evidence that such tradeoffs exist in practice.

1. INTRODUCTION

The race to exascale is rapidly changing supercomputer architecture designs. Shrinking circuit sizes and a growing push toward heterogeneous architectures is yielding systems with processors with many cores, sometimes differing vastly in their capabilities. From a user's standpoint, these changes fundamentally alter the way one interacts with these systems. System resiliency, which traditionally was "free," will no longer be so – lower voltage, a larger number of elements within a node, and elements' shrinking feature sizes is expected to decrease the mean time between failures [29]. Adding extra logic into the hardware to address the resiliency issue only takes up valuable chip real estate; the burden of making sure the application ran to a successful and correct completion may be shifted—at a performance/energy price—to the software.

Another challenge the new architecture designs expose is the power wall problem. As an example, [29] recommends the power wall for exascale systems at 20MW, a limit that is already being flirted with by current-generation petaflop systems[*]. Hardware architects are consequently working closely with application scientists to design systems that can deliver more flops per Watt. Hardware-based solutions alone cannot, however, address all the different stress scenarios that software phases might put on hardware. Part of the solution has to come from the software side as well; these solutions can be addressed by *autotuning.* Autotuning can be defined as a systematic process of navigating the space defined by the software and hardware parameters that impact a metric related to the performance of the system. Next-generation autotuning strategies should efficiently identify and obtain high-performance code optimizations that can help reduce the power demands of key computational pieces of the scientific applications and carefully orchestrate hardware-provided knobs to reduce the power draw. Exascale systems will also provide massive concurrency; hundreds of millions of cores are projected. Writing an application that can take advantage of the available compute resources will provide substantial challenges to today's high-performance computing (HPC) application developers.

Traditionally, the autotuning problem has been expressed as a single-objective (execution time) minimization problem (see, e.g., [12]). However, given current and projected changes in architecture designs, this formulation of the problem is insufficient for a wide variety of emerging autotuning problems. Execution time will be one among several, possibly competing, system-related metrics (such as system resiliency and energy consumption) that must be optimized. Ramping up the speed of the processor to complete the application execution, for example, can jeopardize system resiliency because the increase in chip temperature can make it more vulnerable to failures. Similarly, launching an application to utilize more cores than its computational phases need, or can exploit, wastes energy. Therefore, a *multi-objective formulation of the autotuning problem is needed.*

Multi-objective optimization concerns the study of optimizing two or more objectives simultaneously. Even if there is

[*] Support for this work was provided through the SciDAC program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract No. DE-AC02-06CH11357.

[*] For example, the Japanese K computer requires 12.6MW of power to achieve 10.5 LINPACK petaflops [5].

a unique optimal (software/hardware) decision when any of the objectives is considered in isolation, there may be an entire set of solutions when the objectives are considered collectively. This set is referred to as a Pareto front (formally described in Section 3) and plays an integral role in a wide variety of decision problems in HPC. Two examples relevant to this paper are:

1. HPC administrators increasingly must balance financial costs associated with energy consumption with the need for users to obtain results in a timely manner. In some cases it may be possible to quantify a price on time, and thereby obtain a single, weighted objective comprised of both energy and time costs. However, such *a priori* weights are typically unknown and minimizing such a single objective does not provide information when these weights (or the price of energy) change. A Pareto front in the time-energy space provides optimal solutions for all possible weights/prices.
2. For hardware design and thermal considerations, power capping—where one must perform a computation while satisfying a specified power limit/budget—is increasingly done. Performance tuning in this context could minimize the single objective of run time subject to a constraint on power. However, such a single-objective optimization will not identify the implications associated with that particular power limit. A 1W increase in this limit could be deemed acceptable if it allowed for a 20% reduction in time. Similarly, a decrease in the power limit could result in a negligible performance loss, and thus placing less thermal stress on the hardware would come at minimal cost. A Pareto front in time-power space provides valuable information on the performance consequences of setting power limits.

Hence, multi-objective optimization studies provide significantly richer insight than do single-objective and constrained optimization approaches. The related work summarized in Section 2 provide further examples where considering several metrics simultaneously is of interest.

In this paper, we contribute a mathematical formulation of the multi-objective performance tuning problem. In Section 4 we bridge the terminologies used by the mathematical optimization and performance-tuning communities for the specific case of time, power, and energy metrics. We establish conditions when problems using these metrics benefit from a multi-objective formulation and when the number of objectives of interest can effectively be reduced.

To illustrate the relationship between tuning decisions and multiple, simultaneous objectives, we consider a set of problems based on common HPC kernels. Section 5 presents decision spaces consisting of different loop optimization techniques (e.g., loop tiling, unrolling, scalar replacement, register tiling), clock frequencies, and parallelization (e.g., thread and node counts). We use these problems to conduct an experimental study on multiple objectives on several novel architectures. To the best of our knowledge, this is the first detailed work on empirical analysis of run time, power, and energy tradeoffs on an Intel Xeon Phi coprocessor (Section 6.1), an Intel Xeon E5530 (Section 6.2), and an IBM Blue Gene/Q (Section 6.3). Our results show that tradeoffs exist in practice under a number of different settings.

Although current architectures only expose a limited set of energy-/power-related parameters (e.g., CPU clock frequency) to the software, we anticipate that exascale architectures may admit a richer set of hardware parameters (e.g., power gating of different hardware components) that have power and energy implications. Therefore, we believe that presenting a framework that shows how tradeoffs can be explored is an important contribution to the HPC community. Furthermore, the existence of these tradeoffs can motivate hardware designers to expose a richer set of knobs to future administrators and software designers. This framework and our analysis is sufficiently general and can be easily extended to incorporate new hardware- and software-based power/energy knobs as they become available.

2. RELATED WORK

Several recent works have examined metrics based on performance and power/energy models. An energy-aware compilation framework was developed in [26]. It can estimate and optimize energy consumption of a given code taking as input the architectural and technological parameters, energy models, and energy/performance constraints. A performance-adaptive algorithm for optical interconnects for HPC was proposed in [28] and used to optimize power consumption, throughput, and latency for various traffic patterns. A multi-objective algorithm based on game theory was proposed in [7] for mapping tasks onto multi-core architectures in order to optimize performance and energy. An integrated architecture-circuit optimization framework was used by [11] to study the tradeoff between energy and performance. The authors showed that voltage scaling plays a crucial role in this tradeoff while the choice of an optimal architecture and circuitry does not have a significant impact. The authors in [39] adopted machine learning techniques to build predictive models for power draw, execution time, and energy usage of computational kernels. A "roofline" model for energy that takes into account algorithm characteristics (e.g., operations, concurrency, and memory traffic) and machine characteristics (time and energy costs per operation or per word of communication) was developed in [15]. Using this model, the authors also analyzed the conditions for tradeoffs between time and energy.

Objectives based on architectural simulations have also been used. A multi-objective exploration of the mapping space of a mesh-based network-on-chip architecture was performed in [10]. Using evolutionary computing techniques, the authors obtained the mappings on a performance-power Pareto front. Performance, power, and resource usage objectives were treated by the design space tool in [24] for the purpose of exploring the vast design space of the Grid ALU Processor and its post-link optimizer.

Closer to the present work are exploratory studies using empirical performance data in conjunction with power or energy. The impact of energy constraints for multithreaded applications on multiprocessor applications was studied in [35] and synchronization-aware algorithms were proposed for saving energy consumption with a user-acceptable loss in speedup. The power monitoring devices PowerMon and PowerMon2 were developed in [14] in order to analyze performance and power tradeoffs. The authors in [31] used a power-aware performance prediction model of hybrid MPI/OpenMP applications to develop a novel algorithm to optimize energy consumption and run time. An automated empirical tuning framework that can be configured to optimize both performance and energy efficiency was proposed in [36]. Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems were investigated in [32] and interactions between power and application performance were explored. The empirical performance tuning tool ActiveHarmony was used in [38] to explore the tradeoff between energy consumption and execution time for computationally intensive kernels. The effects of CPU and network bandwidth tuning from a whole-system-level perspective were analyzed in [30]. In demonstrating opportunities for energy savings, tradeoffs between power and run times were found.

Researchers have also begun to explore search algorithms for multi-objective problems. In addition to execution time, many of these works involve objectives that are simpler to evaluate, such as code size, and none have looked at power or energy objectives. Performance and code size were considered in a multi-objective approach in [21] when an unroll factor was varied. A multi-objective evolutionary algorithm was adopted in [22] to find Pareto-optimal (for combinations of code size, compilation time, and execution time) compiler optimization levels. Evolutionary search algorithms were also used in the adaptive compiler framework [33] to find compiler optimization sequences that minimize the objectives code size, average run time, and worst-case run time. Automated tuning of a just-in-time compiler through multi-objective evolutionary search was performed in [23]. The tuning identified optimization plans that are Pareto-optimal in terms of compilation time and a measure of code quality. Milepost GCC [19] is a self-tuning optimization infrastructure that supports general multi-objective optimization where a user can choose to minimize not only execution time but also code size and compilation time. A multi-objective autotuning framework that adopts differential evolution algorithms as a search methodology was developed in [25]. The authors demonstrated the proposed approach by optimizing run time and parallel efficiency when varying loop tiling and thread-count parameters for parallel codes.

3. MULTI-OBJECTIVE OPTIMIZATION: BACKGROUND AND NOTATION

We consider the multi-objective (sometimes called "multicriteria" [18]) mathematical optimization problem

$$\min_{x \in \mathcal{X}} F(x) = [F_1(x), \ldots, F_p(x)], \tag{1}$$

where $p > 1$ objectives are simultaneously minimized. In this paper, we assume that the n-dimensional decision space $\mathcal{X} \subset \mathbb{R}^n$ is a finite collection of discrete points of size $|\mathcal{X}|$. The assumption of a discrete and finite decision space can be relaxed. We assume that each of the p objectives is bounded from below but can take on the extended value "$+\infty$" (e.g., corresponding to an infeasible code transformation within the space $\mathcal{X}$ or a—ideally, reproducible—runtime failure) and that there is at least one point in the decision space $\mathcal{X}$ at which all p objectives are finite.

Many of the standard properties from single-objective optimization have analogies in the multi-objective setting. For example, objectives f that should be maximized can be brought into the framework (1) by defining $F_i(x) = -f(x)$. Similarly, the units of the component objectives F_i do not matter since the solution set of (1) is invariant to shift and positive-scale transformations[†].

In the case of minimizing a single objective f, the idea of (global) optimality is simple: $\hat{x} \in \mathcal{X}$ is optimal if and only if $f(\hat{x}) \leq f(x)$ for all $x \in \mathcal{X}$. However, for multiple objectives we must alter this notion of optimality. The following definitions are standard in multi-objective mathematical optimization (see, e.g., [18]).

Definition 3.1. *We say that $\underline{F(x) \leq F(y)}$ if $F_i(x) \leq F_i(y)$ for all $i = 1, \ldots, p$, and $F(x) \neq F(y)$; in this case we have that y is dominated by x. We say that a point $x \in \mathcal{X}$ is Pareto optimal for (1), or non-dominated, if there is no $y \in \mathcal{X}$ with $F(y) \leq F(x)$. We denote the set of Pareto-optimal points by $\mathcal{X}^* \subseteq \mathcal{X}$. The set of objective function values of all Pareto-optimal points, $\mathcal{F}^* = \{F(x) : x \in \mathcal{X}^*\}$, is called the Pareto front.*

The concepts introduced in Definition 3.1 are perhaps best illustrated by an example. Figure 1 (left) considers the case when the $p = 2$ objectives of time, F_1, and total power, F_2, are simultaneously minimized. The $F_1 \times F_2$ objective space shown is not to be confused with the decision space $\mathcal{X}$ (which in this example corresponds to parameter values defining loop unrolling and other code transformations, see Section 5). For the examples in Figure 1, we assume that the objective values of every feasible decision $x \in \mathcal{X}$ are shown. The shaded area represents the region in $F_1 \times F_2$ space that is dominated by the point C; all points in this region are inferior to C in both objectives. The set of non-dominated points form the Pareto front $\mathcal{F}^*$.

If the objective F_1 (F_2) is minimized in isolation, then we obtain the point A (B), which necessarily belongs on the Pareto front. Similarly, the minimizers of the single objective $f_\lambda(x) = F_1(x) + (1 - \lambda)F_2(x)$, for $\lambda \in [0, 1]$, corresponding to a convex combination of the objectives, will lie on the Pareto front. However, not all points on the Pareto front necessarily correspond to minimizers of a linear combination of the objectives (e.g., point D in Figure 1 (left)).

Hence, the Pareto front contains significantly richer information than one obtains from single-objective formulations. For example, if one were to minimize time subject to a constraint on power, $F_2(x) \leq \overline{P}$, the Pareto front provides the solution for all possible values of the cap $\overline{P}$. In Figure 1 (left), we see that caps of 260W, 257W, and 254W would result in minimal times of 6s, 6.5s, and 8s, respectively.

In some cases, the multiple objectives may not be competing. For the same decision space $\mathcal{X}$ considered in Figure 1 (left), Figure 1 (right) has a second objective of energy consumption, which is strongly correlated with the objective F_1. In fact, the Pareto front now corresponds to a single point, which simultaneously minimizes both objectives.

[†] The solution set for $\min_x F(x)$ is exactly that for $\min_x\{\alpha + \mathrm{diag}(\beta)F(x)\}$ for any $\alpha \in \mathbb{R}^p$ and any positive $\beta \in \mathbb{R}^p$.

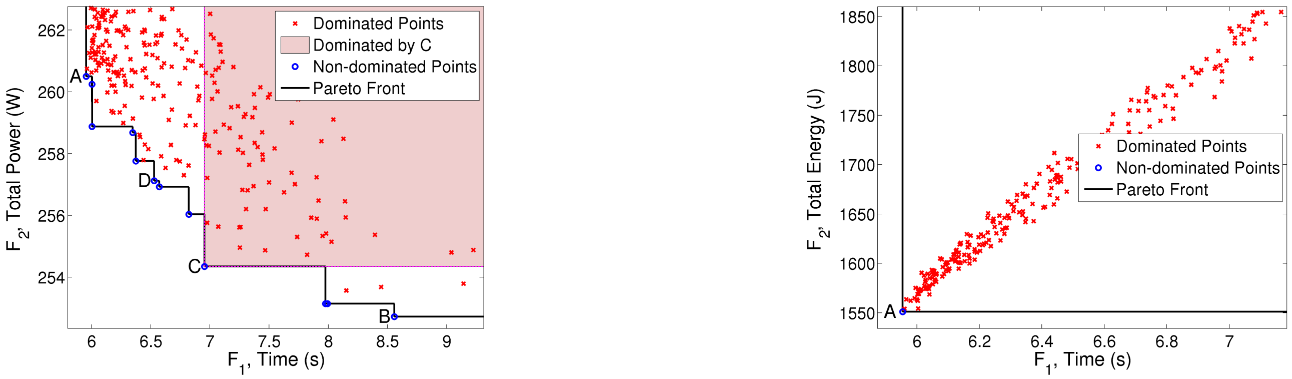

Figure 1: Illustration of Pareto fronts when minimizing two objectives (fdtd kernel, input size 512, Intel Xeon E5530; see Section 6.2). Left: The points `A`, `B`, `C`, and `D` are non-dominated and hence belong to the Pareto front. Right: The Pareto front is a single point, `A`, which dominates all other points.

As evidenced in these examples, only certain regions of the objective space are of interest. Typically, search algorithms for efficiently finding Pareto fronts focus on a hyperrectangle defined by two points formally defined below.

DEFINITION 3.2. *The* <u>*ideal objective point*</u> $F^I = [F_1^I, \ldots, F_p^I]$ *for (1) is defined component-wise by* $F_i^I = \min_{x \in \mathcal{X}} F_i(x)$*. The* <u>*nadir objective point*</u> $F^N = [F_1^N, \ldots, F_p^N]$ *for (1) is defined component-wise by* $F_i^N = \max_{x \in \mathcal{X}^*} F_i(x)$*.*

The ideal point represents the best possible value in each objective. The ideal point can only be attained if the Pareto front consists of a single point as in Figure 1 (right). The nadir point is the extreme point defined by the Pareto front. In the example in Figure 1 (left), the ideal and nadir points are at (5.97s, 252.5W) and (8.57s, 260.5W), respectively. Together, the ideal and nadir point define the range of objective values that a decision maker may encounter if they are interested in all possible optimal tradeoffs.

Before directing our focus on three specific metrics, we note that hard constraints, including those involving an objective of interest, can also be incorporated in (1). We assume that these constraints define the decision space $\mathcal{X}$ and naturally the choice of this decision space can directly affect the objective space, and hence the ideal and nadir points.

4. OPTIMIZATION OF TIME, POWER, AND ENERGY

In this section we focus on the particular *bi-objective* cases where either time and power or time and energy are simultaneously minimized. We could just as easily examine more than two simultaneous objectives in this analysis. However, interpretation/visualization of the empirical results presented in Section 6 would be less straightforward. Furthermore, though our experimental focus is on objectives defined by empirical evaluation, our framework can also include objectives defined by model or simulator evaluation.

For clarity, we will denote the time, power, and energy objectives by T, P, and E, respectively. Since power corresponds to a rate of energy, these two problems (which we can write as $F = [T, P]$ and $F = [T, E]$) are clearly related, with $E = PT$.

There are other properties of these three objectives that one can exploit in their simultaneous optimization. For example, T, P, E are each strictly positive, which allows us to freely multiply/divide by T, P, E without changing inequalities. Similarly, for many problems of interest one can assume that the objective values of two different decision points are different (i.e., for all $x, y \in \mathcal{X}$ with $x \neq y$, $T(x) \neq T(y)$). This property would ensure that there is a one-to-one correspondence between Pareto-optimal decision points $\mathcal{X}^*$ and the Pareto front $\mathcal{F}^*$.

Furthermore, we may have a priori knowledge about the relationship between some decision parameters and some objectives. For example, for many architectures it is safe to assert that power is monotonically increasing in the number of nodes employed. Such relationships can be exploited by both exploratory studies and search algorithms to reduce the number of distinct decision points evaluated.

Because of the relationship between power and energy, we have a simple relationship between the two objective spaces considered here.

DEFINITION 4.1. *Let* $\underline{\mathcal{X}^{*P}} \subseteq \mathcal{X}$ *denote the set of Pareto-optimal points for* $F = [T, P]$ *and let* $\underline{\mathcal{X}^{*E}} \subseteq \mathcal{X}$ *denote the set of Pareto optimal points for* $F = [T, E]$*.*

PROPOSITION 4.2. *All points on the energy-time Pareto front have a corresponding point on the power-time Pareto front:* $\mathcal{X}^{*E} \subseteq \mathcal{X}^{*P}$*.*

PROOF. Let $\hat{x} \in \mathcal{X}^{*E}$ denote a point on the energy-time Pareto front (and hence there is no point $x \in \mathcal{X}$ that dominates $\hat{x}$ for the objectives T and E). Now suppose that $\hat{x} \notin \mathcal{X}^{*P}$, and hence there is some $\tilde{x} \in \mathcal{X}$ that dominates $\hat{x}$. If $T(\tilde{x}) < T(\hat{x})$ and $P(\tilde{x}) \leq P(\hat{x})$, then $E(\tilde{x}) = T(\tilde{x})P(\tilde{x}) < T(\hat{x})P(\hat{x}) = E(\hat{x})$ and hence $\tilde{x}$ is strictly better in both T and P. On the other hand, if $T(\tilde{x}) \leq T(\hat{x})$ and $P(\tilde{x}) < P(\hat{x})$, then $E(\tilde{x}) < E(\hat{x})$. In both cases, $T(\tilde{x}) \leq T(\hat{x})$ and $E(\tilde{x}) < E(\hat{x})$, which contradicts the definition of $\hat{x}$ being non-dominated for the T and E. □

Proposition 4.2 says that the number of non-dominated points for energy-time is bounded by the number of non-dominated points for power-time.

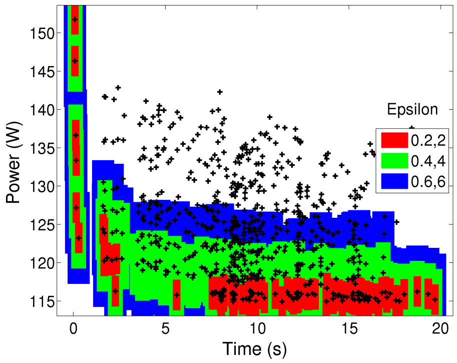

Figure 2: Illustration of the points comprising a relaxed Pareto front for different values of ϵ (SPAPT adi problem, Intel Xeon Phi; see Section 6.1).

Definition 4.3. *Let $x^{(1)} \in \mathcal{X}^{*P}$ denote a non-dominated point on the T-P front that minimizes time: $x^{(1)} \in \arg\min_{x \in \mathcal{X}^{*P}} T(x)$ (where the inclusion is done in case there is not a unique minimizer).*

Proposition 4.4. *A necessary condition for $x \in \mathcal{X}$ to be a non-dominated point on the T-E Pareto front is that*

$$P(x) \leq \frac{P(x^{(1)})T(x^{(1)})}{T(x)}. \quad (2)$$

Proof. By the definition of $x^{(1)}$, $T(x^{(1)}) \leq T(x)$ for all $x \in \mathcal{X}$. Hence, $x \in \mathcal{X}$ can only hope to be on the T-E Pareto front if $E(x) \leq E(x^{(1)})$, which can be rewritten as (2) since $T(x) > 0$ for all $x \in \mathcal{X}$. □

Of course, many necessary bounds exist in addition to (2), but (2) is especially convenient because it provides a convenient bound that requires only a minimizer of a single objective (time). Furthermore, it offers a mathematical relationship for the conditions needed in order for the energy-time Pareto front to be comprised of more than one point. Clearly this inequality does not hold for the example in Figure 1.

Proposition 4.4 can also be used to look at the effect of idle power. If we decompose the power into a constant idle power and a varying difference above idle power, $P(x) = P_I + \Delta P(x)$, then (2) is equivalent to

$$\Delta P(x^{(1)})T(x^{(1)}) - \Delta P(x)T(x) \geq \Big(T(x) - T(x^{(1)})\Big) P_I. \quad (3)$$

A necessary condition for (3) is that the power savings must outpace the product of idle power and relative slow-down,

$$P(x^{(1)}) - P(x) \geq \frac{T(x) - T(x^{(1)})}{T(x^{(1)})} P_I.$$

Hence, for fixed times $T(x)$ and $T(x^{(1)})$, it becomes more unlikely that tradeoffs exist as the idle power P_I grows (since there's always an upper bound to peak available power).

For many time-power-energy multi-objective problems, one may need to acknowledge the measurement error in each objective. Assuming that there is fixed error margin $\epsilon_i \geq 0$ for the ith objective, if $F_i(x)$ is within ϵ_i of $F_i(y)$, then we cannot say that x is truly better than y (or vice versa) with respect to the objective F_i. The notion of non-dominance in Definition 3.1 would thus need to be modified so that x dominates y if $F_i(x) + \epsilon_i \leq F_i(y)$ for all $i = 1, \ldots, p$, and $F(x) \neq F(y)$.

As a result, one would arrive at a *relaxed Pareto front* that potentially consists of a cloud of points. This is illustrated in Figure 2 for different multiples of the measurement error margin ($\epsilon_1 = .2$s,$\epsilon_2 = 2$W). In practice, one often knows what the ϵ_i should be. For example, we know what the measurement resolution of power and time are for each of our experiments; see the measurement descriptions in Section 6.

To simplify the presentation, we follow the convention in Definition 3.1 (which takes $\epsilon_i = 0$ for $i = 1, \ldots, p$) for the results reported in Section 6.

5. PROBLEM SETS & DECISION SPACES

We now describe the set of problems, consisting of HPC kernels from SPAPT [13], TORCH [27], and CSPARSE [16], and the proxy application `miniFE` [20], that we used for our empirical multi-objective study. We also describe the code transformation framework that we utilize to generate variants with different flavors of compiler optimizations.

We consider problems from the SPAPT suite [13]. Each search problem in SPAPT is a specific combination of a kernel, an input size, a set of tunable decision parameters, a feasible set of possible parameter values, and a default/initial configuration of these parameters for use by search algorithms. These problems are implemented in an annotation-based language that can be readily processed by Orio [34]. The tunable decision parameters are loop unroll/jamming, cache tiling, register tiling, scalar replacement, array copy optimization, loop vectorization, and multi-core parallelization using OpenMP. The kernels in SPAPT are grouped into four groups: elementary dense linear algebra kernels, dense linear algebra solver kernels, stencil code kernels, and elementary statistical computing kernels. In this work we consider problems from three of these groups: matrix-matrix multiplication (`mm`), matrix transpose and vector multiplication (`atax`), and triangular matrix operations (`trmm`) from the basic dense linear algebra kernels; bi-conjugate gradient (`bicgkernel`) and `lu` decomposition kernels from the dense linear algebra solver kernels; and matrix subtraction, multiplication, and division (`adi`), 1-D Jacobi computation (`jacobi`), finite-difference time domain (`fdtd`), and matrix factorization (`seidel`) kernels from the stencil code kernels.

To evaluate a set of points in the SPAPT decision space (which can be further extended to include different compiler optimization parameters), we must use a source-to-source transformation framework. We use Orio [34], an extensible and portable software framework for empirical performance tuning. It takes an Orio-annotated C or Fortran implementation of a problem as input, generates multiple transformed code variants of the annotated code, empirically evaluates the performance of the generated codes, and has the ability to select the best-performing code variant using some various search algorithms. Orio annotations consist of semantic comments that encode the computation. A separate tuning specification contains various parameterized performance-tuning directives and sizes of inputs to consider. We refer the reader to [34] for a detailed account on annotation parsing and code generation schemes in Orio.

On multi-core architectures, larger core counts reduces the ratio of peak memory bandwidth to peak floating-point performance. To analyze such behavior, we include two bandwidth-limited problems: a sparse matrix multiplication kernel and a quick sort kernel that sorts n items in $O(n\ log\ n)$ time. The reference implementation of the sparse matrix multiplication kernel is based on CSPARSE, a concise sparse matrix package in C [16], and takes sparse matrix triplets as input. For the quick sort kernel, we use the implementation from the TORCH Computational Reference Kernels [27], a collection of core problems in scientific computing. While in the sparse matrix multiplication kernel, the number of nonzero elements in the matrix lead to floating-point operations, the quick sort kernel performs only comparisons without any significant floating-point operations.

Finally, for large-scale multi-node experiments, we use a proxy application from the Mantevo project, which was designed to explore the capabilities of emerging architectures [20]. `miniFE` is a finite element mini-application that implements kernels representative of unstructured, implicit finite-element applications. It assembles a sparse linear system from a steady-state heat conduction problem on a brick-shaped domain of linear, 8-node hex elements. It then solves the linear-system using a simple (un-preconditioned) conjugate gradient (CG) algorithm. Thus the kernels that `miniFE` contains are: computation of element-operators (diffusion matrix, source vector), assembly (scattering element-operators into sparse matrices and vectors), sparse matrix-vector products (during the CG solve), and vector operations (level-1 BLAS: axpy, dot, norm). Running `miniFE` with a fixed set of dimensions and varying the number of MPI ranks is a commonly used strong scaling test.

6. EXPERIMENTAL RESULTS

We now summarize the findings from our empirical evaluations on three markedly different platforms. The Intel Xeon Phi's Many Integrated Core (MIC) architecture serves as a platform that allows us to explore the tradeoffs among concurrency, power, and performance on nodes with many simple cores, a characteristic that we anticipate will be increasingly common in next-generation large-scale systems. The Intel Xeon E5530 architecture allows us to explore the tradeoffs among power, energy, and performance in a current-generation architecture. The availability of clock frequency scaling on the Xeon E5530 allows us to enrich our decision space $\mathcal{X}$ (see Section 3) with hardware-provided, power-related knobs. Our measurement setup on the Xeon E5530 also provides us with more detailed power measurement capabilities. IBM's BG/Q was chosen as a way to demonstrate our framework's applicability on a vastly different processor architecture and also to explore the tradeoffs among concurrency, power, and performance on a large, multi-nodal scale.

6.1 Intel Xeon Phi

The experiments described in this section are carried out on a first-generation Intel Xeon Phi coprocessor (based on the Intel Many Integrated Core (MIC) architecture) [3], consisting of 60 standard cores clocked at 1090 MHz and with full cache coherency across all cores. Each core offers four-way simultaneous multithreading (SMT) and 512-bit-wide SIMD vectors, which corresponds to 8 double-precision or 16 single-precision floating-point numbers. Each core has a fully coherent 32KB L1 instruction cache, a 32KB L1 data cache, and a 512KB unified L2 cache. The coprocessor card contains 8GB of memory, and is connected via PCI Express bus to a Westmere host running CentoOS 6.3 and with 64GB of host main memory.

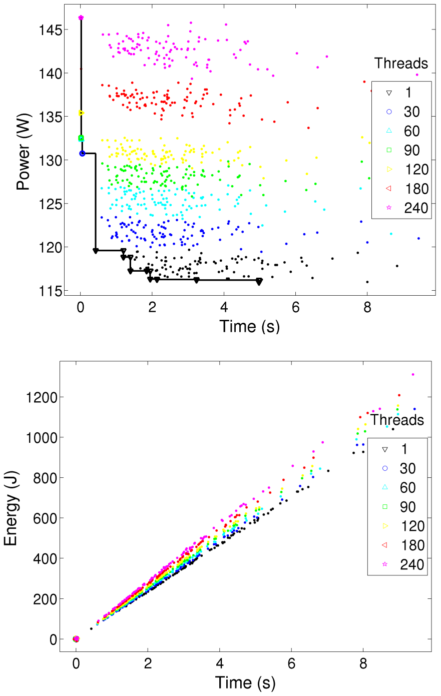

Figure 3: Power, energy, and time for the fdtd SPAPT kernel on Intel Xeon Phi (includes both thread count and code transformation variants).

6.1.1 Setup and Measurement

For power measurement, we relied on the system management utility `micsmc` (v. 4346-16) designed for monitoring and managing Xeon Phi coprocessors. Currently, `micsmc` has a time resolution of 0.5 seconds and power measurement resolution of 1W. The `icc` compiler (version 13.0.0 20120731), with `-mmic` (for native MIC libraries) and `-O3` optimization flags, was used to compile the code variants.

We configure each variant to run k times, where k is selected (separately for each kernel) so that the total run time is at least 50 seconds. Let $r_1(x), \ldots, r_k(x)$ denote a sequence of k run times for the variant x and let $(t_1(x), p_1(x)), \ldots, (t_m(x), p_m(x))$ denote a time-stamped sequence of power measurements obtained from the `micsmc` utility. To calculate power draw for the variant, we consider all power readings $(t_i(x), p_i(x))$ with $r_2(x) \leq t_i(x) \leq 50$ (with $r_1(x)$ omitted to remove any cold-cache effect and the time needed for memory allocation on the card). A 10-second sleep interval in between two successive executions ensures that the processor returns to a normal temperature and power state.

6.1.2 Results

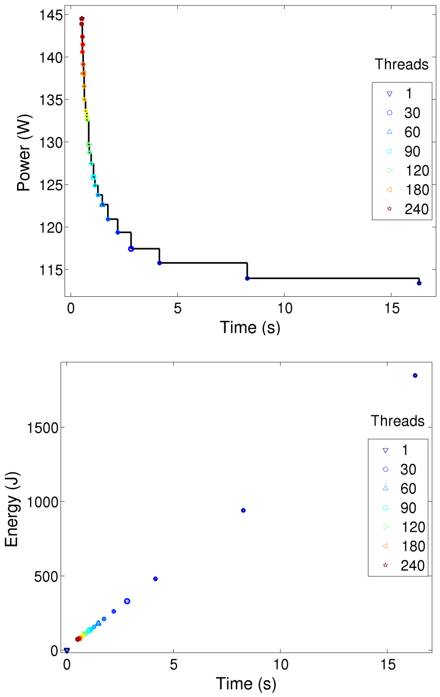

Figure 4: Power, energy, and time for the sparse matrix multiplication kernel on Intel Xeon Phi.

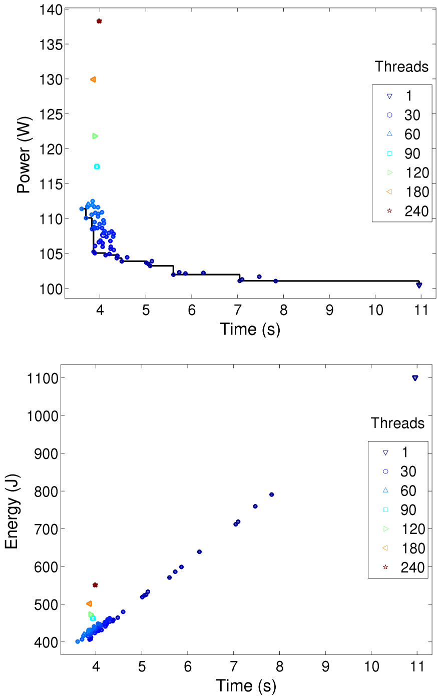

Figure 5: Power, energy, and time for the quick sort kernel on Intel Xeon Phi.

Figure 3 shows the results of obtained on the SPAPT problem `fdtd` (with an input size of 500 × 500). In these plots, we show the average run time, average power, and average energy required by the code variants. The results show a clear tradeoff between run time and power and the number of threads. The number of threads adopted has the largest impact on the power draw whereas the code transformation decisions have the largest impact on run time. We observe that the code variants are clustered based on the number of threads. The power draw increases by approximately 5W with an increase of 30 threads. The corresponding energy plot does not show a tradeoff; it exhibits a race-to-idle condition [8]. Similar trends were seen for other SPAPT problems.

When there is no activity, the coprocessor enters into a complete idle state (PC-state), where it has an efficient power management module to save power and energy by power gating [4]. Currently, the power draw we observe is approximately 60W. However, after transitioning from an idle state to the normal operating state, we observe high idle power (currently between 80W and 90W). Consequently, even a small run time reduction results in significant energy savings. We note that some previous works (e.g., [37, 9]) subtract idle power from the power drawn during the normal operating state in order to only consider the increase in the power draw that can be attributed to a given workload's execution. Our figures show the view from a system operator's perspective, and take into account the total system power (idle and workload computation power).

Next we focus on the sparse matrix multiplication kernel with the input `trdheim`, a large, sparse matrix from the UFL sparse matrix collection [17] with 1,935,324 nonzeros. Other inputs tested (including `std1_Jac3_db`, `biplane-9`, and `t3dl` from [17]), produced similar results. We study the impact of varying the number of threads (concurrency) on run time, power, and energy. Figure 4 shows the Pareto front. Although there is a tradeoff between run time and power, we can observe race-to-idle behavior when it comes to energy efficiency. This can be due to a number of architectural specializations of the Intel Xeon Phi to improve bandwidth [3]. The aggregate bandwidth of L1 and L2 caches are approximately 15 and 7 times faster than the aggregate memory bandwidth, respectively. A 16-stream hardware prefetcher is used to improve the cache hits. It uses a special instruction called "streaming store" that allows the cores to write an entire cache line without reading it first. The interconnect has a 64-byte wide data block ring to support the high bandwidth requirement. Finally, the memory controllers are symmetrically interleaved around the ring to provide a uniform access pattern, which eventually increases the bandwidth response.

Figure 5 shows the results of the quick sort kernel on an input size (the number of random integers to sort) of 10^7. We see a similar trend except that the variants with larger thread counts are slightly slower and thus less energy efficient.

The results from Intel Xeon Phi show that for compute-limited kernels, the use of large core counts results in significant performance benefits with respect to both time and energy. Nevertheless, power is a limiting factor. Due to the effective high-bandwidth memory subsystem, the bandwidth-limited kernels also exhibit a similar trend. It is interesting to note that, in all our Intel Xeon Phi experiments, we observe that the maximum power is between 140W and 145W, irrespective of the type of kernel tested. The average power draw is determined by the number of threads used rather than the type of computations. This observation underscores the importance of developing workload-aware parallelism schemes for the next-generation systems with many cores, so that one uses only the number of cores (or threads) that the workload can actually exploit.

6.2 Intel Xeon E5530

We now describe our results on an Intel Xeon E5530 workstation with two quad-core processors. Each core has its own 32KB L1 cache and 256KB L2 cache; each of the quad-core processors has a shared 8MB L3 cache (for a total of 16MB of L3 for the 8 cores). The processors can be clocked at 1.60, 1.73, 1.86, 2.00, 2.13, 2.26, or 2.39 GHz. Processor clock frequency is changed using the `cpufreq-utils` package [1] that is available with many popular Linux distributions.

6.2.1 Setup and Measurement

Component-level (CPUs and DIMMs) power measurements are collected using a PowerMon2 apparatus [14]. PowerMon2 is a hardware and software framework designed to obtain fine-grained (up to 1024 samples per second) current and voltage measurements for different components of a target system (e.g., CPUs, memory subsystem, disks, GPUs). We measure the system-level power draw using the WattsUp Pro power meter [6]. The power meter is a fairly inexpensive device, costing less than $150 at the time of this writing. Although the device is easy to use, it provides relatively coarse-grained measurements, roughly one reading per second. We implemented a command-line interface on top of the WattsUp driver to monitor and calculate the overall energy usage of an application.

Since we can only measure system level power at 1-second granularity, we configure the main computational loops to run k times, where k is selected (separately for each kernel/input) so that the total run time at the highest CPU frequency is more than five seconds. This ensures that we collect a sufficient number of power readings that can be attributed to the main computation of the kernels. The execution time reported in the paper is for these k iterations of the computation kernel. A post-processing step sweeps through the data to attribute portions of the power measurements to the actual kernel loops. These power measurements are then averaged to determine the power draw for a single execution. To account for the unavoidable noise in this empirical data collection process, we measure each variant three times. The execution time and the power draw reported here are averages of these three runs.

Here we discuss results for the `fdtd`, `jacobi`, and `bicgkernel` SPAPT kernels. However, for `fdtd`, we selected two different input sizes: 512 × 512 (henceforth referenced as `fdtd512`) and 4096 × 4096 (`fdtd4096`). The selection decision was driven by our desire to ensure that we have test cases that stress the CPU and memory subsystem in different ways. Indeed, the last level cache misses per instruction for the base SPAPT case (no transformations) ranges from 1.8×10^{-4} for `bicgkernel` (making it a very compute-bound kernel) to 0.03 for `fdtd4096` (making it a memory-bound kernel).

The code transformations applied to the kernels and the transformation spaces are taken as in [13]. However, we supplement the SPAPT decision spaces with a CPU clock frequency parameter. For each of the kernels, we select 300 (a number chosen simply to limit the time required for data collection while being relatively representative of the large SPAPT transformation spaces) randomly selected variants from the code transformation space. Each of these variants are evaluated on all available clock frequencies.

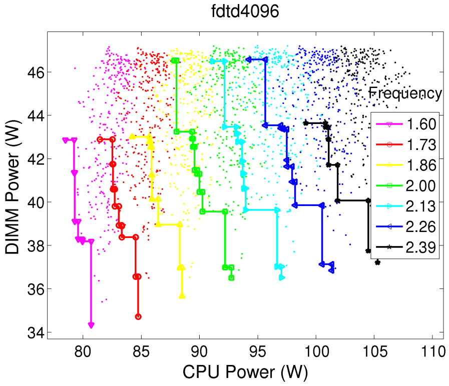

Figure 8: Pareto fronts (for each clock frequency) on Intel Xeon E5530 for component-level power draws.

6.2.2 Results

Figure 6 shows the Pareto fronts for the objectives time and total system power (as measured at the wall). The first observation that demonstrates the richness of the decision space is that, for a given hardware frequency parameter, the power range for the code variants is rather large. Tradeoffs between time and system-level power draw are evident. The power draw is lower for slower clock speeds, but this comes with a slow-down of the computation. Especially interesting is that the Pareto fronts show that there are cases where one can reduce the power draw and not impact the performance substantially. Such results/behavior should be of high interest to co-design centers designing power-limited hardware targeted to specific types of computations.

We can also examine particular transformation variants. Figure 7 shows the energy and time for the five highest-performing (as measured at the fastest clock rate) variants. This figure shows some interesting tradeoff decisions that we can explore. For example, for variant `v1` of the memory-bound `fdtd4096` kernel, we see that we can trade 0.8% loss in performance with 7.5% decrease in the energy consumption by running the kernel at the lowest frequency. The energy savings amount is not as significant for the compute-bound `bicgkernel`, where one can trade 1.2% loss in performance with 2.8% decrease in the energy consumption by running variant `v1` at clock frequency 2.12GHz.

Figure 8 shows the Pareto fronts for each clock frequency for component-level power draws of the `fdtd4096` kernel. When we analyze each of the fronts for different clock frequencies in isolation, we see a clear tradeoff between DIMM and CPU power draws for different code variants. We attribute this behavior intuitively to the optimizations that impact data motion. Code variants that have better data motion behavior reduce the stress on DIMMs thereby lowering the DIMM power. At the same time, better data motion leads to more compute work for the CPU, thereby raising its power de-

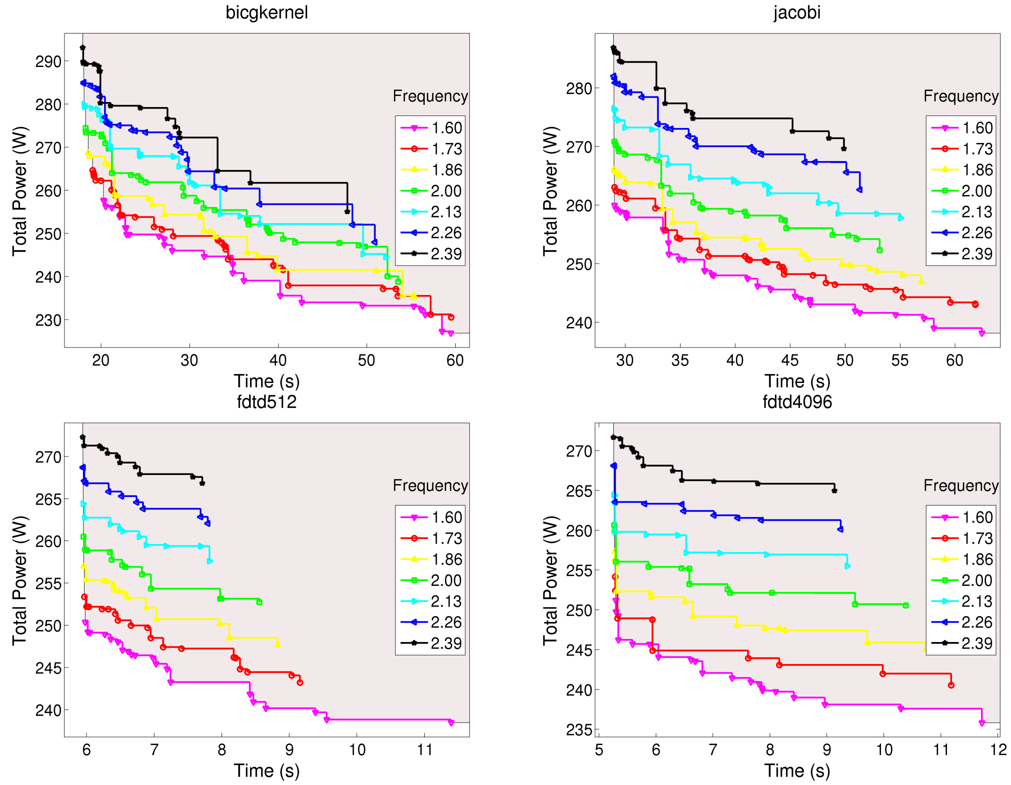

Figure 6: Pareto fronts (for each clock frequency) for SPAPT kernels on Intel Xeon E5530 for the objectives time and total system power. The shaded area shows the Pareto front across all frequencies.

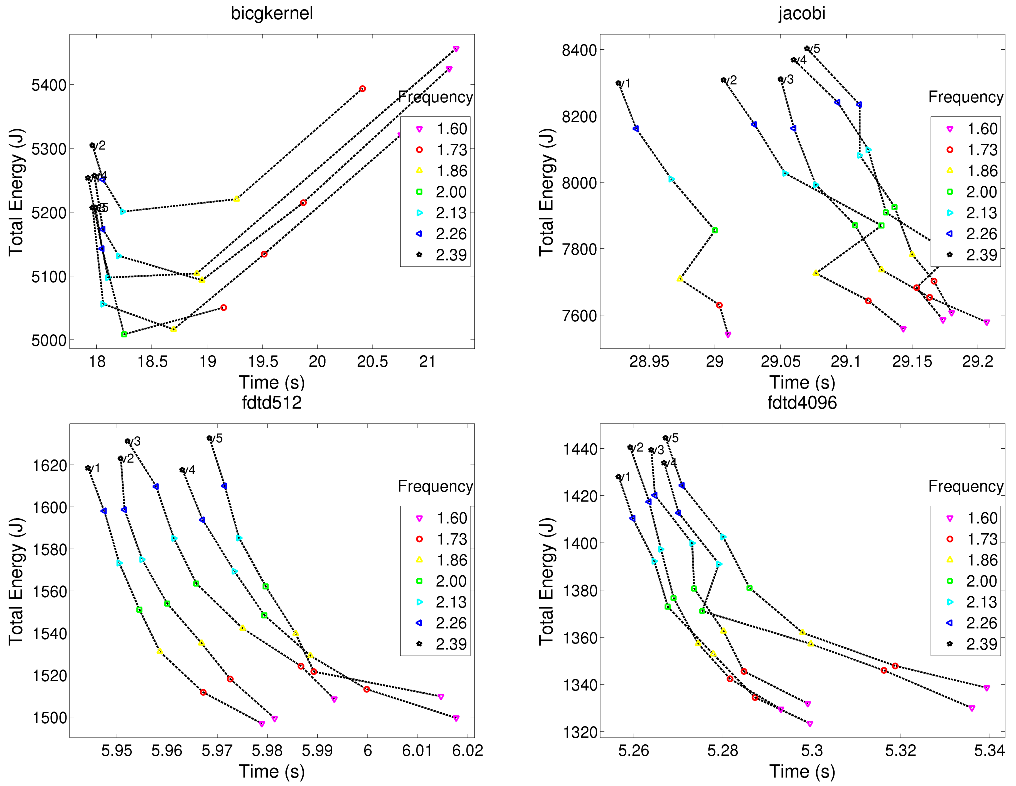

Figure 7: Energy and time on Intel Xeon E5530 for the five highest-performing variants (v1–v5) from the SPAPT transformation space. The curves illustrate the tradeoff behavior as clock frequency is changed.

mand. Such tradeoffs are of interest in studies for future architectures where one may consider constraining CPU draw (e.g., for thermal/fault considerations) and/or DIMM draw (e.g., as a proxy for effective memory footprint or simulator of memory-starved systems).

6.3 Vesta IBM Blue Gene/Q

Vesta is a developmental platform for Mira, a 10-petaflop IBM Blue Gene/Q supercomputer [2] at Argonne. Vesta's architecture is the same as Mira's except that it has two compute racks (compared to the Mira's 48 racks). A rack has 32 node boards, each of which holds 32 compute cards. Each compute card comprises 16 compute cores of 1600 MHz PowerPC A2 processors with 16GB RAM (1GB/core). In total, Vesta has 2,048 nodes (32,768 compute cores). The nodes are connected via a proprietary 5-D torus network. The compute nodes are water-cooled for thermal efficiency and run on CNK, a proprietary, lightweight kernel that minimizes OS noise.

6.3.1 *Setup and Measurement*

For the power measurements in BG/Q, we use a power profiling code that periodically samples power draw [40]. Due to cabling and control system limitations, the code requires a minimum partition size of 128 nodes, which spans 4 node boards. The profiler code runs one thread on each node board and records the power on all the domains every 0.25 seconds along with a time stamp. We refer the reader to [40] for further details on the power profiling in BG/Q.

We set the input size (controlling the box domain from which a finite-element problem is assembled and solved) of `miniFE` to $n_x = n_y = n_z = 1000$. We considered a decision space with four parameters: two generic parameters that control the scaling behavior and two application-specific parameters. The generic parameters are the number of nodes ({128, 256, 512, 1024}) and the number of threads per core (either 8 (one thread every other core) or 16 (one thread per core)). The two `miniFE` specific parameters are the percentage of unbalance in the decomposition ({5, 10, 20, 30, 40, 50, 60, 70, 80, 90}) and a boolean decision parameter ({Yes, No}) that controls whether matrix-vector products are performed with overlapping communication and computation. In total, we had 160 code variants for the experimental analysis.

6.3.2 *Results*

The results in Figure 9 show that there are tradeoffs between time to completion and both power and energy. As expected, increasing the node count decreases the time to completion but increases the power draw. In addition to the workload power, the significant increase in the power draw can be attributed to the fact that each node board consumes an idle power of ≈1500W [40]. The node count of 1024 uses 32 node boards but 128 uses only 4 node boards. Concerning energy, the best parameter configuration within each node count provides a tradeoff between time to completion and energy consumption. However, within a given node count, the fastest code variant consumes the least energy.

7. CONCLUSIONS AND OUTLOOK

In this paper we have provided a formalism for multi-objective optimization studies of broad applicability in auto-tuning, architecture design, and other areas of HPC. With a focus on time, power, and energy, we illustrated that a multi-objective analysis provides richer insight than do constrained and single-objective formulations. We have also contributed a significant empirical study, spanning a diverse set of platforms, power measurement technologies, kernels, and decision spaces. Our findings showed that in some settings objectives are strictly correlated and there is a single, "ideal" decision point; in others, significant tradeoffs exist.

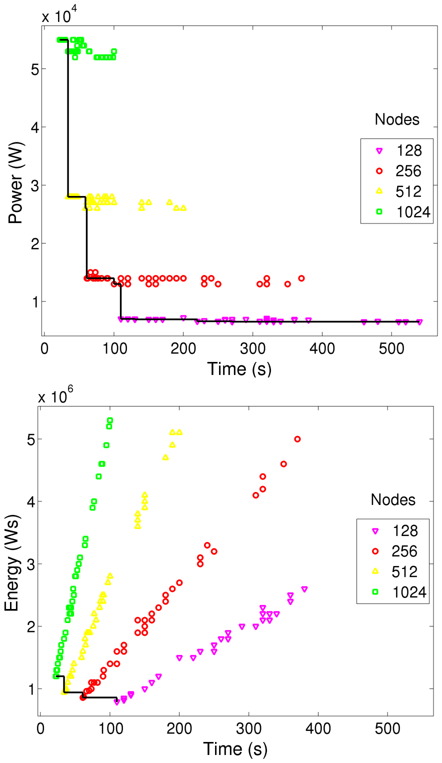

Figure 9: Power, energy, and time for `miniFE` on BG/Q.

For exhaustive evaluation of a decision space, multi-objective analysis is straightforward, since it requires only that the multiple objectives be recorded. However, for more targeted processes (such as when searching for decisions on the Pareto front), the analysis can be more expensive than single-objective analysis.

Future work includes characterizing settings where empirical tradeoffs agree with those predicted by models (e.g., the roofline work in [15]) and where relationships between objectives are not as well understood. There are significant opportunities for studying the tradeoffs among additional objectives; we especially mention resiliency since its relationship to power-/temperature-based objectives is expected to be a prime concern in future extreme-scale systems [29].

Acknowledgments

Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research. This research used the computational resources of the Argonne Leadership Computing Facility under a Director's discretionary allocation. We thank Paul Hovland for valuable discussions and Kazutomo Yoshii for insights on the power monitoring systems of BG/Q and Intel Xeon Phi and valuable feedback.

8. REFERENCES

[1] CPU Freq. Scaling. `https://wiki.archlinux.org/index.php/Cpufrequtils`.

[2] IBM System Blue Gene Solution - Overview. `http://www-03.ibm.com/systems/technicalcomputing/solutions/bluegene/`.

[3] Intel Xeon Phi Coprocessor - the Architecture. `http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner`.

[4] Intel Xeon Phi Coprocessor System Software Developers Guide. `http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-system-software-developers-guide`.

[5] TOP500 List. `http://www.top500.org`. June 2012 Report.

[6] WattsUp? Meters. `https://www.wattsupmeters.com/`.

[7] I. Ahmad, S. Ranka, and S. U. Khan. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–6. IEEE, 2008.

[8] S. Albers and A. Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1266–1285. SIAM, 2012.

[9] P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo, and E. S. Quintana-Orti. Saving energy in the LU factorization with partial pivoting on multi-core processors. In *20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 353–358. IEEE, 2012.

[10] G. Ascia, V. Catania, and M. Palesi. Multi-objective mapping for mesh-based NoC architectures. In *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 182–187. ACM, 2004.

[11] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz. Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 26–36. ACM, 2010.

[12] P. Balaprakash, S. M. Wild, and P. D. Hovland. Can search algorithms save large-scale automatic performance tuning? *Procedia Computer Science*, 4:2136–2145, 2011.

[13] P. Balaprakash, S. M. Wild, and B. Norris. SPAPT: Search problems in automatic performance tuning. *Procedia Computer Science*, 9:1959–1968, 2012.

[14] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield. PowerMon: Fine-grained and integrated power monitoring for commodity computer systems. In *IEEE SoutheastCon 2010*, pages 479–484, 2010.

[15] J. W. Choi and R. Vuduc. A roofline model of energy. Technical Report GT-CSE-12-01, Georgia Institute of Technology, School of Computational Science and Engineering, December 2012.

[16] T. A. Davis. *Direct methods for sparse linear systems*, volume 2. SIAM, 2006.

[17] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, Dec. 2011.

[18] M. Ehrgott. *Multicriteria Optimization*. Springer-Verlag, 2nd edition, 2005.

[19] G. Fursin, Y. Kashnikov, A. W. Memon, Z. Chamski, O. Temam, M. Namolaru, E. Yom-Tov, B. Mendelson, A. Zaks, E. Courtois, et al. Milepost gcc: Machine learning enabled self-tuning compiler. *International Journal of Parallel Programming*, 39(3):296–327, 2011.

[20] M. A. Heroux, D. W. Doerer, P. S. Crozier, and J. M. Willenbring. Improving performance via mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, September 2009.

[21] K. Heydemann and F. Bodin. Iterative compilation for two antagonistic criteria: Application to code size and performance. In *Proceedings of the 4th Workshop on Optimizations for DSP and Embedded Systems*, 2006.

[22] K. Hoste and L. Eeckhout. Cole: Compiler optimization level exploration. In *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, pages 165–174. ACM, 2008.

[23] K. Hoste, A. Georges, and L. Eeckhout. Automated just-in-time compiler tuning. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 62–72. ACM, 2010.

[24] R. Jahr, T. Ungerer, H. Calborean, and L. Vintan. Automatic multi-objective optimization of parameters for hardware and code optimizations. In *International Conference on High Performance Computing and Simulation (HPCS)*, pages 308–316. IEEE, 2011.

[25] H. Jordan, P. Thoman, J. J. Durillo, S. Pellegrini, P. Gschwandtner, T. Fahringer, and H. Moritsch. A multi-objective auto-tuning framework for parallel codes. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 10:1–10:12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[26] I. Kadayif, M. Kandemir, N. Vijaykrishnan, M. Irwin, and A. Sivasubramaniam. EAC: A compiler framework for high-level energy estimation and optimization. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pages 436–442. IEEE, 2002.

[27] A. Kaiser, S. Williams, K. Madduri, K. Ibrahim, D. Bailey, J. Demmel, and E. Strohmaier. TORCH computational reference kernels: A testbed for computer science research. Technical Report UCB/EECS-2010-144, EECS Department, University of California, Berkeley, December 2010.

[28] A. Kodi and A. Louri. Performance adaptive power-aware reconfigurable optical interconnects for high-performance computing (HPC) systems. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC)*, pages 1–12, 2007.

[29] P. Kogge. The tops in flops. *IEEE Spectrum*, 48(2):48–54, 2011.

[30] J. H. Laros III. Measuring and tuning energy efficiency on large scale high performance computing platforms. Technical Report SAND2011-5702, Sandia

National Laboratories, August 2011.

[31] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos. Hybrid MPI/OpenMP power-aware computing. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12. IEEE, 2010.

[32] C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, and K. Cameron. Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems. *International Journal of High Performance Computing Applications*, 25(3):342–350, 2011.

[33] P. Lokuciejewski, S. Plazar, H. Falk, P. Marwedel, and L. Thiele. Multi-objective exploration of compiler optimizations for real-time systems. In *13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, pages 115–122, 2010.

[34] B. Norris, A. Hartono, and W. Gropp. Annotations for productivity and performance portability. In *Petascale Computing: Algorithms and Applications*, Computational Science, pages 443–462. Chapman & Hall / CRC Press, 2007.

[35] S. Park, W. Jiang, Y. Zhou, and S. Adve. Managing energy-performance tradeoffs for multithreaded applications on multiprocessor architectures. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 169–180. ACM, 2007.

[36] S. F. Rahman, J. Guo, and Q. Yi. Automated empirical tuning of scientific codes for performance and power consumption. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, pages 107–116. ACM, 2011.

[37] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh. Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 230–238. ACM, 2006.

[38] A. Tiwari, M. Laurenzano, L. Carrington, and A. Snavely. Auto-tuning for energy usage in scientific applications. In *Euro-Par 2011: Parallel Processing Workshops*, pages 178–187. Springer, 2012.

[39] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely. Modeling power and energy usage of HPC kernels. In *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pages 990–998. IEEE, 2012.

[40] K. Yoshii, K. Iskra, R. Gupta, P. Beckman, V. Vishwanath, C. Yu, and S. Coghlan. Evaluating power-monitoring capabilities on IBM Blue Gene/P and Blue Gene/Q. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 36–44. IEEE, 2012.